EXHIBIT 31

Power Script
= = Documentation = =

Document Version 2.16

Power.com

1. Revision History

Version	Date	Author	Description
1.0	4/25/2007	Fábio Passos	Document created.
2.0	4/29/2007	João Gilberto Rezende Magalhães	Layout adjustment and new commands added.
2.1	5/31/2007	João Gilberto Rezende Magalhães	Document modifications.
2.2	06/08/2007	João Gilberto Rezende Magalhães	Document modifications.
2.3	06/18/2007	João Gilberto Rezende Magalhães	Document modifications.
2.4	06/19/2007	Raphael Souza de Oliveira	Document modifications.
2.5	06/20/2007	João Gilberto Rezende Magalhães	Variables section – cookie added.
2.6	06/27/2007	Raphael Souza de Oliveira	Variables section – Changing the filter's behavior
2.7	06/28/2007	Raphael Souza de Oliveira	Execute-If section – Added less-than, less-equal-than, greater-than, greater-equal-than
2.8	07/02/2007	João Gilberto Rezende Magalhães	Catch and Procedure
2.9	07/06/2007	Raphael Souza de Oliveira	Execute-If section – Changed less-than, less-equal-than, greater-than, greater-equal-than. For-each section – Added collection-length-var attribute. New section – Creating a Data type variable.
2.10	07/10/2007	João Gilberto Rezende Magalhães	Functions and FOR resources.
2.11	07/12/2007	Raphael Souza de Oliveira	Creating a Data type variable section – Added datetime-culture and datetime-format-output attributes. New section – "XML" Rule
2.12	07/13/2007	João Gilberto Rezende Magalhães	New attribute to "read" rule
2.13	07/13/2007	João Gilberto Rezende Magalhães	New ARRAYLEN function.
2.14	07/19/2007	João Gilberto Rezende Magalhães	New ENCODE and DECODE functions.
2.15	09/03/2007	João Gilberto Rezende Magalhães	New rule: Plugin
2.16	01/24/2008	João Gilberto Rezende Magalhães	Manipulation of Resources (*.resx)
2.16	08/25/2008	Danilo José Pereira Delgado	Modifying filter behavior section – Included range and direction properties Making a request section – Included post-mode and max-tries properties.

Power.com

Developer Manual

Power.com

1. Power Script

PowerScript is a tool to browse and extract data from HTML pages. Script execution creates automated browsing on Internet sites and extracts the desired information. In addition to browsing rules it's possible to add additional rules to the spider, such as database manipulation, file saving, sending e-mails, among others.

The script programmer needs to be knowledgeable in reading and interpreting HTML documents with a WebSurfer profile.

1.1. Data Extraction

The basic purpose of the script is to extract data from any HTML document in a relatively simple way for the programmer. For example, it's possible to extract a text delimited by identifiers, extract a specific attribute in an HTML node, obtain the selected item in a combo, and much more.

2. Context

The context is the information obtained by regular browsing of a website, such as cookies and session indicators. The same context may be shared by various scripts. Therefore, if a script, for example, logs a user into a certain site, the resulting context may be reused in another script, which will be read as if the user had already logged in. The context name is defined in the login script and uses the extension .context.xml.

The context may be used in several ways by the application – it can be stored as a text file, may be saved in a database, or even stored by the web service Elisa.

3. Reference

3.1. Script Structure

The structure of the script is defined in two parts: Context and Rules

```
<processpage>
      <context name="somename" />
      <rules>
            <rule action="SOMEACTION">
                  <!--
                  Other actions
                  -->
            </rule>
      </rules>
</processpage>
```

3.2. Context

Define a name for the context resulting from browsing. If a certain script generates session information, for example, this same context name may be used in other scripts and all the session information may be resused.

The context may be saved in a text file (with the extension ".context.xml") or in the memory. See API documentation for more information.

3.2.1. Attributes

Attribute	Req.	Possible Values	Description
name	Yes	[Text]	Contains the context name. It is not necessary to add an extension, and it must not contain spaces.
reset	No	true false	If the value is **true**, the existing context will be erased and a new one resulting from script browsing will be created.

Example:

```
<context name="somecontext" />
```

3.3. The Rules

All processing and execution of the script are based on RULES. Each rule executes a specific action. A rule may be contained inside another. The script will then obey the following order:

```
<processpage>
      <rule action="first">
      </rule>
      <rule action="second">
            <rule action="third">
                  <rule action="fourth">
                  </rule>
            </rule>
            <rule action="fifth">
            </rule>
      </rule>
      <rule action="sixth">
      </rule>
</processpage>
```

3.4. Rule "setvar"

The "setvar" rule makes it possible for a user to define one or more variables inside the script.

```
<rule action="setvar">
      <var name="somename" value="somevalue" />
      <var name="somename2" value="somevalue2" />
</rule>
```

3.4.1. Variables

The variable is an important piece in the script definition, because all the extracted values are stored in it.

A variable inside a script has some characteristics that are important to note:

- Once a variable is defined it has its global scope and is valid until the end of script execution.
- Every variable created in the script is the array type, even if it has just one element stored. A number of arguments support an array as a parameter, so that it causes recursive execution of the command for EACH element of the array. This is a very important characteristic, because it can greatly simplify script execution and creation.
- The content of a variable is always text.
- To use the content of a variable inside an argument it's necessary to place the name inside the symbol "#". Ex.:

```
<var name="somename" value="somevalue" />
<var name="somename2" value="#somename#" />
```

3.4.2. Attribution of values

Attribute	Req.	Possible Values	Description
name	Yes	[Text]	Contains the variable name. If ONLY this attribute is defined, the content of the HTML obtained by the "read" rule is stored under this name. **EVERY VARIABLE REQUIRES AN ATTRIBUTE NAME.**
value	No	[Text]	Defines the value that will be stored in the variable under the name "name". If this attribute is not passed, the value automatically assumed will be the HTML obtained by the last READ rule executed.
once	No	true false	If "true", this variable will be defined just once. Its behavior is that of a constant. But this attribute is particularly useful for defining external attributes passed by the API. It's important to stress that the definitions by API are defined BEFORE script execution. Because the argument works this way, it will NOT change the value if it was passed initially by API. Default: false.

3.4.3. Obtain the values of an attribute

We can extract the attributes of a specific HTML element.

For example: To extract all the hyperlinks of a specific HTML document, we can use the following statement:

```
<var name="somename" attribute="href" of-tag="a" />
```

Attribute	Req.	Possible Values	Description
attribute	Yes	[Text]	Defines the attribute to be extracted. If more than one attribute is found, an array will be automatically generated.
of-tag	Yes	[Text]	Defines the tag from which to extract an attribute.

3.4.4. Obtain the text of a tag

It's also possible to extract the text within a tag. For example:

```
<var name="somename" text-element="true" of-tag="a" />
```

You will obtain a text from the tag “A” as listed below:

```
<a href="somelink">THIS TEXT</a>
```

Attribute	Req.	Possible Values	Description
text-element	Yes	true	Informs that it wishes to extract the text of a tag from a specific node. For example: `<a href="">TEXT</a>`
of-tag	Yes	[Text]	Defines the tag from which to extract the text.

3.4.5. Obtain a text between two values

It's possible to obtain the text between two values existing in an HTML document. It’s important to note that the tex is not necessarily inside a tag.

```
<var name="somename" after="sometext" before="sometext2" />
```

Attribute	Req.	Possible Values	Description
after	Yes	[Text]	Informs the first limit of the text to be extracted
before	Yes	[Text]	Informs the last limit of the text to be extracted
include-limit	No	true false	If true, the content from after and before are added to the value obtained. (Default: false)
revert	No	true false	If true, the after and before will be applied to the inverted text.

3.4.6. Modifying filter behavior

In the rules for variable extraction and definition, it's possible to add filters that define if the value should or should not be obtained, as well as basic actions that modify the value defined. These attributes may also be combined with each other.

Attribute	Possible Values	Description
substring	[Start];[Size]	Allows extraction of a substring starting from the [start] position and size [size]
reset	true false	If the value is **true**, the existing context will be erased and a new one resulting from script browsing will be created.
distinct	true false	If the value is true, only unique elements will be added to the array.
trim	true false	If the value is true, it will remove all the extra spaces from each element of the array.
index	[1-10000]	Allows only one element of the array defined by index to be selected. The first element starts with number 1.
add	true false	If the value is true, it will add elements to the array instead of replacing it.
ignore-empty	true false	If the value is true, only elements with value will be added to the array.
substring-reverse	[Start];[Size]	Allows extraction of a substring from back to front. The [start] = “0” position corresponds to the last character and size [size] corresponds to the number of characters which should be obtained.
replace	text1|text2	Substitutes text1 for text2. The | symbol separates text1 from text2.
fix-amp	true false	Changes all the & symbols to &.
html-chars	encode decode	Makes it possible to have the HTML encode/decode the value of the array.
url-chars	encode decode	Makes it possible to have the URL encode/decode the value of the array.
contain	[Text]	Only adds the value to the vector if it CONTAINS the text.
not-contain	[Text]	Only adds the value to the vector if it DOES NOT CONTAIN the text.
encrypt-key	[Text]	Allows ciphering/deciphering a text according to the key provided.
encrypt-url-chars	encode decode	Same for the url-chars attribute, but it is applied after the encrypt-key.
encrypt-html-chars	encode	Same for the attribute html-chars, but it is applied

join	decode separator	after the encrypt-key. Transforms an array of n positions to one of 1 position, with the items from the previous array separated by the SEPARATOR.
split	separator	Transforms an array of 1 position to one of n positions, where each item of the new one will be the text contained in the interval in each occurrence of the SEPARATOR.
pad	[Type];[Size]; [PaddingChar]	Fills a string up to the desired [Size] size with the character informed in [PaddingChar] according to the [Type]; L for the left or R for the right.
range	[Start];[Size]	Cuts the array in the interval defined between [Start] plus [Size].
direction	input output	Determines if the value of the variable is entry or output.

3.4.7. Creating a variable of the Date type

Variables that contain Date information can be created. The attributes below can be combined amongst each other.

Attribute	Possible Values	Description
datetime-culture	http://msdn2.microsoft. com/en-us/library/ system.globalization. cultureinfo(VS.80).aspx "Culture Names and Identifiers" section, "Culture Name" column.	Culture which will be used to convert the entry value (attribute value) in a date object and to format the variable output value.
datetime-format-output	http://msdn2.microsoft. com/en-us/library/ system.globalization. datetimeformatinfo(VS.80).aspx	Format which the date will have in the output variable.
day	[Number]	Informs the day that will be used to build the date; if not informed, the "current date" day will be used or the day of the date sent by the value attribute with the valid format according to the informed culture.
month	[Number]	Informs the month that will be used to build the date; if not informed, the "current date" month will be used from the date sent by the attribute value with the valid format according to the informed culture.
year	[Number]	Informs the year that will be used to build the date; if not informed, the "current date" year will be used

		from the date sent by the attribute value with the valid format according to the informed culture.
hour	[Number]	Informs the hour that will be used to build the date; if not informed, the “current date” hour will be used from the date sent by the attribute value with the valid format according to the informed culture.
minute	[Number]	Informs the minute that will be used to build the date; if not informed, the “current date” minute will be used from the date sent by the attribute value with the valid format according to the informed culture.
second	[Number]	Informs the second that will be used to build the date; if not informed, the “current date” second will be used from the date sent by the attribute value with the valid format according to the informed culture.
time-interval	[Number]	Informs the value which will be added or subtracted from the Date.
time-interval-type	Year Month Day Hour Minute Second	Informs the part of the date from which the value informed by the time-interval will be added or subtracted.

3.4.8. Reading a value from the configuration file

Allows the value of a key in ELISA or in APP.CONFIG to be attributed to a variable.

```
<var name="variable" load-config="KEYINELISA"/>
```

3.4.9. Special variables

The spider automatically creates the variables containing the cookie values. To do so, it adds a variable for each cookie. The name of this variable is automatically defined to: “**cookie.NAMEOFCOOKIE**”.

A special variable called “**cookie.$list$**” is also created, which contains the list of cookies defined in the varaibles. This contains the cookies separated by ;.

3.4.10. Use of functions and expressions

The scrips allows pre-defined functions to be executed in two ways:

- Adding an evalue="true" clause to the varaible
- Creating an expression between #[e]#

In both cases, the following functions can be executed:

Function	Description
Mod(arg1; arg2)	The remainder of the division between arg1 and arg2
div(arg1; arg2)	The quotient of the division between arg1 and arg2
add(arg1; arg2)	The sum of arg1 and arg2 parcels
sub(arg1; arg2)	The subtraction between arg1 and arg2
Mult(arg1; arg2)	The multiplication of arg1 by arg2
random(arg1; arg2)	A random number between arg1 and arg2
len(arg1)	The size of the string in arg1
lenarray(arg1)	The size of the array represented by the variable in arg1. Note that no special character should be used; only the name of the variable should be used.
encode(html|url; arg2)	Encodes the HTML or URL of arg2
decode(html|url; arg2)	Encodes the HTML or URL of arg2

Example:

```
<rule action="setvar">
      <var name="v" value="19" />
      <var name="v1" value="The value is: #[ mod(13; 5) ]#" />
      <var name="v2" value="The mod #v# by 5 is: #[ mod(%v%; 5) ]#" />
      <var name="v3" value="mult(mod(13; 5); 10)" evaluate="true" />

      <var name="a" value="30" add="true" />
      <var name="a" value="45" add="true" />
      <var name="a1" value="lenarray(a)" evaluate="true" />

      <var name="e1" value="decode(html, %v2%)" evaluate="true" />
      <var name="e2" value="http://x/x.aspx?#[encode(url, %v1%)]#" />
</rule>
```

3.4.11. Examples of variable utilization

In the example below, the value *www.powerscrap.com* is attributed to *var_value1*.

```
<var name="var_value1" value="www.powerscrap.com"/>
```

If you are reading the value from another variable it's necessary to use the (#) in the extremities. The value of *var_value2* will be the value contained in *var_value1* which is www.powerscrap.com.

```
<var name="var_value2" value="#var_value1#"/>
```

In this example an array will be created with 3 positions under the name *var_add* with the values 1,2,3 in the respective.

```
<var name="var_add" value="1"/>
<var name="var_add" value="2" add="true"/>
<var name="var_add" value="3" add="true"/>
```

If, after the example above, an attribute is used without the *add="true"* the array will go back to having a position, in this case the value will be 4.

```
<var name="var_add" value="4"/>
```

In the example below, a *var_index* variable is created with just one position with the value of position 2 of the *var_add* vector from the example above. If the index were not used, the whole array would be copied to *var_index*.

```
<var name="var_index" name="#var_add#" index="2"/>
```

In the example below, the title of the page is captured. Remember that the "<" symbol uses the form < and the symbol ">" uses the form >.

```
<var name="var2" after="&lt;title&gt;" before="&lt;/title&gt;"/>
```

This is the classic case of creating a variable with more than one position in the array, in the case of the *after* and *before* text not being unique on the page.

In this case, the value is read from var_value1 which contains the value *www.powerscrap.com*, and after the *after* and *before* are applied the value *powerscrap* is attributed.

```
<var name="var3" value="#var_value1#" after="www." before=".com"/>
```

In this case, the value of the *rel* attribute is captured from all the *links* tags; that is, if there is more than one *link* tag, an array will be generated with more than one position.

```
<var name="var4" attribute="rel" of-tag="link" />
```

This can also be used to capture the value of the *value* property, in this case the value of *var5* will be *www.power.com*.

```
<var name="var5" value="&lt;link href='www.power.com' /&gt;"
attribute="href" of-tag="link"/>
```

In this example, the value of the variable will be *power*.

```
<var name="var6" value="www.powerscrap.com" substring="4;5"/>
```

In the example below, the value of the variable will be *spider* without spaces.

```
<var name="var7" value="  spider  " trim="true" />
```

In the example below, the value of the variable will be *www.power.com*.

```
<var name="var8" value="www.orkut.com" replace="orkut|power"/>
```

In this example, the value of the variable will be the title of the page since it already contains the word *orkut*.

```
<var name="var9" after="&lt;title&gt;" before="&lt;/title&gt;"
contain="orkut"/>
```

In this example the value of the variable will be empty, since the title of the page does not contain the word *power*.

```
<var name="var10" after="&lt;title&gt;" before="&lt;/title&gt;"
contain="power"/>
```

Suppose a page contains the following text:

```
<form method="post" action="/Scrapbook.aspx?uid=9384887058608207927
&amp;pageSize=30" autocomplete="off">

<var name="var_fixamp" attribute="true" of-tag="form" fix-amp="true"/>
```

The extracted text will be */Scrapbook.aspx?uid=938488705860820792****&*** *pageSize=30*

If *fix-amp* is not used the extracted will be */Scrapbook.aspx?uid=938488705860820792****&****pageSize=30*.

3.5. Rule "read"

This rule makes it possible to access a page. The result of this page will be in a buffer which can be manipulated through the <VAR> tag. The default request method is GET. If there are any <POST> tags in any part inside the rule RULE, the system will automatically do a POST instead of a GET. This rule does not allow rules within rules.

Ex.:

```
<!-- Example of GET requisition -->
<rule action="read" param="http://www.orkut.com/Edit.aspx?uid=#id#">
</rule>

<!-- Example of POST requisition -->
<rule action="read" param="http://www.orkut.com/Edit.aspx?uid=#id#">
      <post name="somename" value="somevalue" />
</rule>
```

It's important to note that in the "param" attribute, many combinations of variables may be passed on. The script will try to make a combination of values and will execute a REQUEST for each element of the resulting array.

3.5.1. Making a requisition

Attribute	Req.	Possible Values	Description
action	Yes	read	Determines that the action is reading.
stop	Yes	[url]	Determines the URL which will be requested.

			It's important to note that after reading, the content accessed will be available on the buffer to be collected by the <VAR> directive.
delay	No	[miliseconds]	Informs the waiting time BEFORE starting said request.
valid-exists	No	[text]	This attribute validates if the content of the page obtained by the request contains the defined text. If it does not contain the text, an error is created. If the “code-error” attribute has been defined, the system will create an error message defined in this attribute.
valid-not-exists	No	[text]	This attribute validates if the content of the page obtained by the request does NOT contain the defined text. If it contains the text, an error is created. If the “code-error” attribute has been defined, the system will create an error message defined in this attribute.
code-erro	Não	[texto]	Error message for the valid-exists and valid-not-exists attributes.

3.5.2. Making a FOR loop.

It's possible to create a loop with a pre-set start and finish.

Attribute	Req.	Possible Values	Description
action	Yes	read	Determines that the action is reading.
param	Yes	[url]	Determines the URL which will be requested. It's important to note that after reading, the content accessed will be available on the buffer to be collected by the <VAR> directive.
for	Yes	[variable]	Determines the variable which will receive the value resulting from the loop from START to END. Note that the variable may NOT be an expression with (#). This variable may be used in a parameter or in any rule inside this rule.
start	Yes	[number]	Determines the initial loop value.
end	No	[number]	Determines the end of the loop. It's not mandatory, because this value may be defined after the first reading. The name of the variable automatically created is the name of the variable followed by _end. To define the value in the loop the instruction <VAR> must contain the ONCE clause.
step	Não	[number]	Determines the increment of the FOR loop from START to END.

3.5.3. The Post statement

By defining a “post” instruction within a “read” rule, the method of requisition is automatically keyed to POST.

Attribute	Req.	Possible Values	Description
Name	Yes	[Text]	Contains the name of the POST parameter.
Value	Yes	[Text]	Contains the value to be sent.
index	No	[1-10000]	If a value is sent through the INDEX, the post will only be an element referred to by position. If not, the POST will be done with ALL the elements of the array.
not-empty	No	true false	If true, it will only create an entry in POST if the element is not empty. The default is always to post any element.

3.5.4. Sending a cookie

A cookie may be sent thruogh a web requisition.

```
<cookie name="cookiename" value="cookievalue" />
```

Attribute	Req.	Possible Values	Description
Name	Yes	[Text]	Contains the cookie name.
Value	Yes	[Text]	Contains the cookie value.
domain	No	[Text]	The cookie domain.
path	No	[Text]	The cookie path.

3.5.5. Content Validation

In addition to validation in the definition of the rule, it's possible to add a directive called <validation>. This needs to be added within the <rule action=”read”>clause. The major difference is that this clause supports EXPRESSIONS (substitutions of variables) while the clause inside the rule definition validates the entire document.

```
<!-- The expression #AUX# must not have "forgotpass" -->
<validation value="#aux#" not-exists="forgotpass" code-error="5002"/>

<!-- The expression #AUX# must have pageerror -->
<validation value="#aux#" exists="pagaerror" code-error="5003"/>
```

3.5.6. Modifying requisition header

It's possible to modify the requisition header by adding the following code inside the <rule action=”read”>clause:

```
<httpheader>
      <header name="" value="" />
      <header name="" value="" />
      <header name="" value="" />
</httpheader>
```

3.6. "Database" rule

Through this rule, it's possible to interact with the database executing any DML script. Initially, it's necessary to set the following options in the configuration file (app.config):

```
<!--
  DATABASE CONNECTION
  Object (including namespace) to connect to database
  Example: System.Data.SqlClient.SqlConnection
-->
<add key="spider.DATABASECONNECTION"
value="System.Data.SqlClient.SqlConnection" />

<!--
  DATABASE ASSEMBLY
  Assembly name (without .dll) it contains Object Connection
  Example: System.Data
-->
<add key="spider.DATABASEASSEMBLY" value="System.Data" />

<!--
  CONNECTION STRING
  name represents the name of connection string.
-->
<add key="spider.CONNECTIONSTRING.name"
value="Server=name;Database=name;User Id=xyz;Password=" />
```

The <database> clause defines the connection and type of transaction, while the <query> clauses define the type of consultation to be done.

3.6.1. The connection definition inside the script

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "database"
connection	Yes	[Text]	Indicates which connection name the queries will use. The connection names are defined inside the configuration file (app.config).
transaction	No	true false	If true, indicates that the execution of all queries will be transactional.

3.6.2. Executing searches

There are 5 different types of searches: free text, scalar, dynamic intelligence, dynamic update, dynamic delete. This rule does not allow rules within rules.

```
<rule action="database" connection="string" transaction="true" >

  <!-- This type of search defines a Free SQL -->
  <query sql="text" index="1">
    insert into table (field1, field2) values ('#var#', '#var2#');
  </query>

  <!-- This type of search defines a Free SQL but the return must
mandatorily be just ONE VALUE. This value will be stored in “variable”
-->
  <query sql="scale" index="1" result="variable">
    select count(*) from table
  </query>
  <!--
      ATENTION: The Execute-on uses LOTS of machine resources. Use
only if necessary. -->
  <query sql="dynupdate" table="name" index="1" execute-
on="expression" sql-where=" expressao " >
    <field name="field" key="true" value="#var#" default="NULL"
type="string|number|date" />
    <field name="field" value="#var#" default="NULL"
type="string|number|date" />
  </query>
  <query sql="dyninsert" table="name" index="1" execute-
on="expression" >
    <field name="field" key="true" value="#var#" default="NULL"
type="string|number|date" />
    <field name="field" value="#var#" default="NULL"
type="string|number|date" />
  </query>
  <query sql="dynintelligent" table="name" index="1" execute-
on="expression" sql-where=" expression " >
    <field name="fieldkey" key="true" value="#var#" default="NULL"
type="string|number|date" />
    <field name="field" value="#var#" default="NULL"
type="string|number|date" depend-table="xxx" depend-field="zzz" />
  </query>
</rule>
```

3.7. “Savefile” Rule

This rule makes it possible to save content from a URL or content from a variable in the file system. This rule does not allow rules within rules.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: “savefile”
url	No	[Text]	Array which indicates the URL of the files to be saved on the disk. NOTE: If this parameter is not sent, the rule automatically understands that it is to use the InnerXml expression inside <rule></rule>
path	No	[Text]	Indicates the default path to save documents. If omitted, uses the current path or the FULLNAME defined in “filename”
filename	No	[Text]	Indicates the full name of the document. If omitted,

			assumes the full name of the URL.
overwrite	No	true false	If defined as TRUE, overwrites existing file; if defined as FALSE, adds a new file with a counter.

3.8. "E-mail" Rule

This rule makes it possible to send an e-mail through the script. The message to be sent is the InnerXML expression inside the"email" rule. This rule does not allow rules within rules.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "email"
smtpserver	Yes	[Text]	Informs the name of the SMTP server which will send the messages.
authuser	No	[Text]	If the SMTP server requires authentication, the user should be sent.
authpass	No	[Text]	If the SMTP server requires authentication, the username and password should be sent.
from	Yes	[Text]	Sender e-mail
to	Yes	[Text]	Recipient e-mail. Supports arrays.
subject	Yes	[Text]	E-mail subject.

3.9. "Delay" Rule

This rule pauses script execution for the time informed. This rule does not allow rules within rules.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "delay"
time	Yes	[milisecs]	Time in milliseconds of stop.

3.10. "Executeif" Rule

This rule limits the execution of a specific block only if a certain condition is met. A condition is based on the result of an expression compared if it is empty; not empty, equal or not equal to the value.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "executeif"
cond	Yes	[Expression]	The result of some variable
type	Yes	equal	The type of comparation to be made.

		not-equal empty not-empty	NOTE: If cond and value are NUMERIC values, the less-than, less-equal-than, greater-than, greater-equal-than will do a numeric to numeric comparison; otherwise, they will do a string to string comparison.
value	No	[text]	This field is only used when the type of comparison is equal or not-equal.

3.11. "For-each" Rule

This rule makes it possible to go through EVERY element of a variable of the type of array and generates a repetition with these values.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "for-each"
collection	Yes	#var#	The expression which contains the array.
item	Yes	[variable]	Name of the variable which will receive the unit element referring to the loop in collection.
indexvar	No	[variable]	Name of the variable which will contain as many loops as executed.
collection-length-var	No	[variable]	Name of the variable which will contain the collection size in question.

3.12. Rule "For"

The "for" rule makes it possible to create a loop a certain known number of times.

Attribute	Req.	Possible Values	Description
action	Yes	[Text]	Name of action: "for"
var	Yes	[Text]	Name of the variable which will be created for the loop.
start	Yes	[Whole]	First value of the loop to be attributed in "var"
end	Yes	[Whole]	Last value of the loop to be attributed in "var"
step	No	[Whole]	Increment of the "var" variable"

```
<rule action="for" var="i" start="1" end="10" step="1">
      <rule ......>
      </rule>
</rule>
```

3.13. "Throw" Rule

Allows an exception to be created with a specific message.

```
<rule action="throw" message="exception message" />
```

3.14. “Call” (Procedures) Rule

With PowerScript, the code can be organized into procedures to reuse the code. In this case, the “call” rule should be used and the name of the procedure should be informed, as in the example below.

One interesting characteristic is that the procedure parameter of this rule accepts an array, such that multiple calls can be made in a single rule.

Note: The node procedure must be inside the “processpage” in order to be found.

```
<processpage>
      <context name="to-be-set" />
      <rules>
            .
            .
            <rule action="call" procedure="name_procedure" />
            .
            .
      </rules>

      <procedure name="name_procedure">
            <rule ... >
            </rule>
      </procedure>
</processpage>
```

3.15. “XML” Rule

This rule allows the creation of the xml type variables. To do so, simply follow the example shown below:

```
<processpage>
  <context name="not-be-set" />
  <rules>
    .
    .
    .

    <rule action="xml" xml-output-var="variable" version="version"
encondig="encoding" standalone="standalone">
      <element name="name_element_1">
        <attribute name="name_attribute" value="attribute value"/>
        <element name="name_element_2">
          <element name="name_element_3" value="value 3" />
          <rule action="for-each" ... >
            <rule action="xml" xml-output-var="variable">
              <element name="name_element_4">
                <element name="name_element_5" value="value 5"/>
              </element>
            </rule>
          </rule>
```

```
        </element>
      </element>
    </rule>
  </rules>
</processpage>
```

In “variable” output we have:

```
<?xml version="version" encoding="encoding" standalone="standalone" ?>
<name_element_1 name_atribut="attribute value">
  <name_element_2>
    <name_element_3>value 3</name_element_3>
    <name_element_4>
      <name_element_5>value 5</name_element_5>
    </name_element_4>
    <name_element_4>
      <name_element_5>value 5</name_element_5>
    </name_element_4>
  </name_element_2>
</name_element_1>
```

Attribute	Req.	Possible Values	Description
version	No	[Text]	Version of the xml to be created; if not informed, the system assumes “1.0”.
enconding	No	[Text]	Encoding which will be used by the xml to be created; if not informed, assumes “utf-8”.
standalone	No	yes no	Informs if the xml is or is not standalone; if not informed, the xml does not use this property.

3.16. “Plugin” rule

This rule allows classes which implement he IPluginScript interface to be incorporated to the script which is being executed.

3.16.1. The Interface:

```
public interface IPluginScript
{
   // Will be executed when a rule is found
   void RuleStart(CookieCollection currentContext,
      NameArrayListCollection variables,
      NameArrayListCollection parameters);

   // Will be executed for each method found within the rule
   void RunMethod(string methodName, string currentBuffer,
      NameArrayListCollection variables,
      NameArrayListCollection parameters);

   // Finalizes the rule processing and returns the modified cookies
   // Returns NULL so as not to change anything.
   CookieCollection RuleEnd();
}
```

3.16.2. A plugin example

```
namespace spider.parser
{
    public class PuglinSample : IPluginScript
    {
        protected CookieCollection _currentContext;

        public void RuleStart(CookieCollection currentContext,
            spider.infra.NameArrayListCollection variables,
            spider.infra.NameArrayListCollection parameters)
        {
            this._currentContext = currentContext;
        }

        public void RunMethod(string methodName, string currentBuffer,
            spider.infra.NameArrayListCollection variables,
            spider.infra.NameArrayListCollection parameters)
        {
            // Do something here.
        }

        public System.Net.CookieCollection RuleEnd()
        {
            return null; // Para alterar o contexto, retorne NULL;
        }
    }
}
```

3.16.3. Example of a script with the plugin

```
<rule action="plugin" class="spider.parser.PuglinSamples"
      assembly="spider" param1="value1"  . . . paramN="valueN">

      <meumetodo1 param1="value1" . . . paramN="valueN" />
      <meumetodo2 param1="value1" . . . paramN="valueN" />
            .
            .
            .
      <meumetodoN param1="value1" . . . paramN="valueN" />

</rule>
```

3.16.4. Parameters of the plugin rule

Attribute	Description
action	Should always be “plugin” to access this functionality.
class	Name of the class, including the NameSpace.
assembly	Name of the assembly which contains the above class. Do not include the DLL. The assembly should stay in the bin folder.

All of the parameters other than these will be sent as an argument of the RuleStart method.

3.16.5. Method parameters

The methods within the rule are executed one by one within the “plugin” rule. The method name and the parameters in this method are sent as an argument of the RunMethod method.

3.17. “Resource” Rule

Upon defining this rule, PowerScript associates one or more resource files (*.resx) to the script and treats it as if it were a new variable. The process uses cache mechanism, and in practice does not add new variables. It only makes their utilization accessible.

```
<rule action="resource" class="NAME" />
```

It’s important to note that the NAME of the resource should be sent without any refernece to the paths, extensions, or culture.

- The path is obtained through Web.Config or the Elisa variable: RESOURCE.PATH
- The culture is obtained through the CurrentUICulture
- The extension is added automatically.

To obtain a key within the resource file, simply treat it like any other variable of the script with the “resource” prefixo. For example:

```
<var name="example" value="#resource.KEY#" />
```

An important observation is that the key should contain only TEXT or NUMBERS (no periods, etc.).

3.17.1. API

```
ResourceHelper rh = new ResourceHelper();
rh.AddResource("nome");
   .
   .
   .
rh.AddResource("nome");


rh.Get("KEY");
```

Power.com

4. Error Treatment (Catch)

Each “RULE” rule and its nested rules may have their block protected by one or more “catch” clauses.

These clauses prevent the error from being passed on to the user. The basic principle is that of treating errors by code validation (validation, valid-exists, valid-not-exists, etc), but it can also treat other errors through its messages.

It’s important to note that the error treatment is valid ONLY for the block within the “READ” rule. The other rules are not protected.

```
<processpage>
      <context name="to-be-set" />
      <rules>
            <rule action="read" param="http://www.uol.com.br">
                  <validation code-error="1000" valid-exists="xx" />
                  .
                  .
                  .
                  <catch code-error="1000">
                        <rule>
                        </rule>
                  </catch>
            </rule>
      </rules>
</processpage>
```

5. Examples

The examples below are a demonstration of how the commands may be grouped to extract more complex data.

5.1. Text Field

The text field is identified by the *input type=”text”* tag, as in the html below:

...
<input type="text" maxlength="40" size="24" name="city" id="city" value="Salvador" />
…

The command below creates the *city* variable with the value of the *value* attribute of the *input* tag which contains the text *"city"* (the double quotes are described as *"* in html).

```
<var name="city" attribute="value" of-tag="input"
contain="&quot;city&quot;" />
```

Power.com

Notes: There should only be one occurrence of an *input* tag containing the text *"city"*, otherwise, an array will be created with more than one position in the *city* variable.

5.2. Combo box

The combo box is identified by the *select* tag, as in the html below:

```
...
<select id="status" name="status">
<option value="0" selected>no answer</option>
<option value="1" >single</option>
<option value="2" >married</option>
<option value="3" >committed</option>
<option value="4" >open marriage</option>
<option value="5" >open relationship</option>
</select>
...
```

The selected item has the text *selected* within the *option* tag.

To get the selected item, the command must be done in two stages. First, attribute the entire select text to a temporary variable and then only after is the selected value obtained.

The command below creates an auxilliary variable with the name *aux* which obtains the text after the *<select* (the less than symbol < is described as < in HTML) and before the *</select* which contains the text "status".

```
<var name="aux" after="&lt;select" before="&lt;/select"
contain="&quot;status&quot;"  />
```

Now that you have the *select* code you want attributed to the *aux* variable, select the *option* which contains the *selected* text. Notice that the *value* attribute is reading the *aux* variable attributed above. The (#) must be used in the extremities to identify the reading of a variable.

```
<var name="relationshipStatusId" value="#aux#" attribute="value" of-
tag="option" contain="selected" />
```

5.3. Option Box

The option box is identified by the *input type=radio* tag, as in the html below:

```
...
<input type=radio class=checkbox id=gender0 name=gender value=1
onchange="validateRequiredRadio(true, 'gender', 2);"/>
<label for=gender0>female</label>
<input type=radio class=checkbox id=gender1 name=gender value=2
onchange="validateRequiredRadio(true, 'gender', 2);"
checked=true/>
<label for=gender1>male</label>
```

…
There is a small problem to get the option selected, because for each option there is a *input type=radio* tag with the same name. So the text of the options must be placed in an auxilliary variable, as in the command below.

```
<var name="aux" after="preceeding text" before="posterior text"  />
```

Preceeding text = a text that is before the options and that is unique in the HTML code.

Posterior text = a text that is after the options and that is unique in the HTML code.

Now, a variable called *genderID* is created which will store the value of the attribute value of the *input type=radio* tag which contains the text *checked=true*;

```
<var name="genderId" value="#aux#"  attribute="value" of-tag="input"
contain="checked=true" />
```

5.4. Check box

The check box is identified by the *input type=”checkbox”* tag, as in the HTML below:

```
...
<input type="checkbox" id="hereFor0" name="hereFor0" class=checkbox
checked="checked" />
<label for="hereFor0">friends</label>
<br>
<input type="checkbox" id="hereFor1" name="hereFor1" class=checkbox /> <label
for="hereFor1">activity partners</label>
<br>
<input type="checkbox" id="hereFor2" name="hereFor2" class=checkbox /> <label
for="hereFor2">business networking</label>
<br>
<input type="checkbox" id="hereFor3" name="hereFor3" class=checkbox />
<label for="hereFor3">dating</label>
…
```

To get the value of the check boxes, the code of each check box must be stored in an auxilliary variable.

```
<var name="aux" after="preceeding text"
before="&quot;hereFor1&quot;"/>
```

preceeding text = a text that is before the check boxes and that is unique in the HTML code.

A variable is created to store if it's checked or not; in this case the variable is *interestedFriends*. A strategy is used to get the value of 0 or 1. The value of the *id* attribute of the *input* tag that contains the *checked* text is retrieved. If it does not contain the *checked* text, the value of the variable is null and when inserted in the database it

may be inserted with the default value 0; if it contains the *checked* text, the value returned will be the *id* which in this case is *heroFor0*, so here a replace is applied in *heroFor0* for 1.

```
<var name="interestedFriends" value="#aux#" attribute="id" of-
tag="input" contain="checked" replace="hereFor0|1"/>
```

To get the other values of the check boxes, continue the same thing in the first instance. It's easier to find a unique text to be the preceeding and posterior text when the check boxes are neither the first nor last.

Example for the second check box:

```
<var name="aux" after="&quot;hereFor0&quot;"
before="&quot;hereFor2&quot;"/>
<var name="interestedPartners" value=":#aux#:" attribute="id" of-
tag="input" contain="checked" replace="hereFor1|1"/>
```

6. Configuration File

The configuration of the spider must be done in the configuration file of your application (App.config). The properties are defined by param key and value.

6.1. Debug Level

This property defines the debug level in which the message generated by the spider in console mode will be shown.

<add key="spider.DEBUGLEVEL" value="0" />

0 = No message, only from errors.
1 = Only rules.
2 = Rules and variables.
3 = All.

6.2. Class of connection with the database.

The name of the namespace of the connection with the database must be defined.

<add key="spider.DATABASECONNECTION"
value="System.Data.SqlClient.SqlConnection" />

The example above is defining the namespace of the connection with the Sql Server.

6.3. Database data assembly

The name of namespace of the database must be defined. The dataset, for example, is in this namespace.

<add key="spider.DATABASEASSEMBLY" value="System.Data" />

6.4. Connection String

Defines a connection string used to connect to the database. The server, database, username and password or if it's security integrated must be defined. More than one connection string may be defined by creating another key with the prefix spider. CONNECTIONSTRING concatenated with an identifier for the connection string.

<add key="spider.CONNECTIONSTRING.name"
value="server=localhost;database=powerclone;integrated security=true;" />